Internet Engineering Task Force (IETF)
Request for Comments: 7661
Obsoletes: 2861
Category: Experimental
ISSN: 2070-1721

G. Fairhurst
A. Sathiaseelan
R. Secchi
University of Aberdeen

October 2015

Updating TCP to Support Rate-Limited Traffic

Abstract

This document provides a mechanism to address issues that arise when TCP is used for traffic that exhibits periods where the sending rate is limited by the application rather than the congestion window. It provides an experimental update to TCP that allows a TCP sender to restart quickly following a rate-limited interval. This method is expected to benefit applications that send rate-limited traffic using TCP while also providing an appropriate response if congestion is experienced.

This document also evaluates the Experimental specification of TCP Congestion Window Validation (CWV) defined in RFC 2861 and concludes that RFC 2861 sought to address important issues but failed to deliver a widely used solution. This document therefore reclassifies the status of RFC 2861 from Experimental to Historic. This document obsoletes RFC 2861.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7661.

Table of Contents

1. Introduction

TCP is used for traffic with a range of application behaviours. The TCP congestion window (cwnd) controls the maximum number of unacknowledged packets/bytes that a TCP flow may have in the network at any time, a value known as the FlightSize [RFC5681]. FlightSize is a measure of the volume of data that is unacknowledged at a specific time. A bulk application will always have data available to transmit. The rate at which it sends is therefore limited by the maximum permitted by the receiver advertised window and the sender congestion window (cwnd). The FlightSize of a bulk flow increases with the cwnd and tracks the volume of data acknowledged in the last Round-Trip Time (RTT).

In contrast, a rate-limited application will experience periods when the sender is either idle or unable to send at the maximum rate permitted by the cwnd. In this case, the volume of data sent (FlightSize) can change significantly from one RTT to another and can be much less than the cwnd. Hence, it is possible that the FlightSize could significantly exceed the recently used capacity. The update in this document targets the operation of TCP in such rate-limited cases.

Standard TCP states that a TCP sender SHOULD set cwnd to no more than the Restart Window (RW) before beginning transmission if the TCP sender has not sent data in an interval exceeding the retransmission timeout, i.e., when an application becomes idle [RFC5681]. [RFC2861] notes that this TCP behaviour was not always observed in current implementations. Experiments confirm this to still be the case (see [Bis08]).

Congestion Window Validation (CWV) [RFC2861] introduced the term "application-limited period" for the time when the sender sends less than is allowed by the congestion or receiver windows. [RFC2861] described a method that improved support for applications that vary their transmission rate, i.e., applications that either have (short) idle periods between transmissions or change the rate at which they send. These applications are characterised by the TCP FlightSize often being less than the cwnd. Many Internet applications exhibit this behaviour, including web browsing, HTTP-based adaptive streaming, applications that support query/response type protocols, network file sharing, and live video transmission. Many such applications currently avoid using long-lived (persistent) TCP connections (e.g., servers that use HTTP/1.1 [RFC7230] typically support persistent HTTP connections but do not enable this by default). Instead, such applications often either use a succession of short TCP transfers or use UDP.

Standard TCP does not impose additional restrictions on the growth of the congestion window when a TCP sender is unable to send at the maximum rate allowed by the cwnd. In this case, the rate-limited sender may grow a cwnd far beyond that corresponding to the current transmit rate, resulting in a value that does not reflect current information about the state of the network path the flow is using. Use of such an invalid cwnd may result in reduced application performance and/or could significantly contribute to network congestion.

[RFC2861] proposed a solution to these issues in an experimental method known as CWV. CWV was intended to help reduce cases where TCP accumulated an invalid (inappropriately large) cwnd. The use and drawbacks of using the CWV algorithm described in RFC 2861 with an application are discussed in Section 2.

Section 3 defines relevant terminology.

Section 4 specifies an alternative to CWV that seeks to address the same issues but does so in a way that is expected to mitigate the impact on an application that varies its sending rate. The updated method applies to the rate-limited conditions (including both application-limited and idle senders).

The goals of this update are:

- To not change the behaviour of a TCP sender that performs bulk transfers that fully use the cwnd.
- To provide a method that co-exists with standard TCP and other flows that use this updated method.
- To reduce transfer latency for applications that change their rate over short intervals of time.
- To avoid a TCP sender growing a large "non-validated" cwnd, when it has not recently sent using this cwnd.
- To remove the incentive for ad hoc application or network stack methods (such as "padding") solely to maintain a large cwnd for future transmission.
- To provide an incentive for the use of long-lived connections rather than a succession of short-lived flows, benefiting both the long-lived flows and other flows sharing capacity with these flows when congestion is encountered.

Section 5 describes the rationale for selecting the safe period to preserve the cwnd.

1.1. Implementation of New CWV

The method specified in Section 4 of this document is a sender-side-only change to the TCP congestion control behaviour of TCP.

The method creates a new protocol state and requires a sender to determine when the cwnd is validated or non-validated to control the entry and exit from this state (see Section 4.3). It defines how a TCP sender manages the growth of the cwnd using the set of rules defined in Section 4.

Implementation of this specification requires an implementor to define a method to measure the available capacity using a set of pipeACK samples. The details of this measurement are implementation-specific. An example is provided in Section 4.5.1, but other methods are permitted. A sender also needs to provide a method to determine when it becomes cwnd-limited. Implementation of this may require consideration of other TCP methods (see Section 4.5.3).

A sender is also recommended to provide a method that controls the maximum burst size (see Section 4.4.2). However, implementors are allowed flexibility in how this method is implemented, and the choice of an appropriate method is expected to depend on the way in which the sender stack implements other TCP methods (such as TCP Segment Offload (TSO)).

1.2. Standards Status of This Document

The document obsoletes the methods described in [RFC2861]. It recommends a set of mechanisms, including the use of pacing during a non-validated period. The updated mechanisms are intended to have a less aggressive congestion impact than would be exhibited by a standard TCP sender.

The specification in this document is classified as "Experimental" pending experience with deployed implementations of the methods.

2. Reviewing Experience with TCP-CWV

[RFC2861] described a simple modification to the TCP congestion control algorithm that decayed the cwnd after the transition to a "sufficiently-long" idle period. This used the slow-start threshold (ssthresh) to save information about the previous value of the congestion window. The approach relaxed the standard TCP behaviour

for an idle session [RFC5681], which was intended to improve application performance. CWV also modified the behaviour when a sender transmitted at a rate less than allowed by cwnd.

[RFC2861] proposed two sets of responses: one after an "application-limited period" and one after an "idle period". Although this distinction was argued, in practice, differentiating the two conditions was found problematic in actual networks (see, e.g., [Bis10]). While this offered predictable performance for long on-off periods (>>1 RTT) or slowly varying rate-based traffic, the performance could be unpredictable for variable-rate traffic and depended both upon whether an accurate RTT had been obtained and the pattern of application traffic relative to the measured RTT.

Many applications can and often do vary their transmission over a wide range of rates. Using [RFC2861], such applications often experienced varying performance, which made it hard for application developers to predict the TCP latency even when using a path with stable network characteristics. We argue that an attempt to classify application behaviour as application-limited or idle is problematic and also inappropriate. This document therefore explicitly avoids trying to differentiate these two cases, instead treating all rate-limited traffic uniformly.

[RFC2861] has been implemented in some mainstream operating systems as the default behaviour [Bis08]. Analysis (e.g., [Bis10] and [Fai12]) has shown that a TCP sender using CWV is able to use available capacity on a shared path after an idle period. This can benefit variable-rate applications, especially over long delay paths, when compared to the slow-start restart specified by standard TCP. However, CWV would only benefit an application if the idle period were less than several Retransmission Timeout (RTO) intervals [RFC6298], since the behaviour would otherwise be the same as for standard TCP, which resets the cwnd to the TCP Restart Window after this period.

To enable better performance for variable-rate applications with TCP, some operating systems have chosen to support non-standard methods, or applications have resorted to "padding" streams by sending dummy data to maintain their sending rate when they have no data to transmit. Although transmitting redundant data across a network path provides good evidence that the path can sustain data at the offered rate, padding also consumes network capacity and reduces the opportunity for congestion-free statistical multiplexing. For variable-rate flows, the benefits of statistical multiplexing can be significant, and it is therefore a goal to find a viable alternative to padding streams.

Experience with [RFC2861] suggests that although the CWV method benefited the network in a rate-limited scenario (reducing the probability of network congestion), the behaviour was too conservative for many common rate-limited applications. This mechanism did not therefore offer the desirable increase in application performance for rate-limited applications, and it is unclear whether applications actually use this mechanism in the general Internet.

Therefore, it was concluded that CWV, as defined in [RFC2861], was often a poor solution for many rate-limited applications. It had the correct motivation but the wrong approach to solving this problem.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The document assumes familiarity with the terminology of TCP congestion control [RFC5681].

The following additional terminology is introduced in this document:

- cwnd-limited: A TCP flow that has sent the maximum number of segments permitted by the cwnd, where the application utilises the allowed sending rate (see Section 4.5.3).

- pipeACK sample: A measure of the volume of data acknowledged by the network within an RTT.

- pipeACK variable: A variable that measures the available capacity using the set of pipeACK samples (see Section 4.2).

- pipeACK Sampling Period: The maximum period that a measured pipeACK sample may influence the pipeACK variable.

- Non-validated phase: The phase where the cwnd reflects a previous measurement of the available path capacity.

- Non-validated period (NVP): The maximum period for which cwnd is preserved in the non-validated phase.

- Rate-limited: A TCP flow that does not consume more than one half of cwnd and hence operates in the non-validated phase. This includes periods when an application is either idle or chooses to send at a rate less than the maximum permitted by the cwnd.

- Validated phase: The phase where the cwnd reflects a current estimate of the available path capacity.

4. A New Congestion Window Validation Method

This section proposes an update to the TCP congestion control behaviour during a rate-limited interval. This new method intentionally does not differentiate between times when the sender has become idle or chooses to send at a rate less than the maximum allowed by the cwnd.

In the non-validated phase, the capacity used by an application can be less than that allowed by the TCP cwnd. This update allows an application to preserve a recently used cwnd while in the non-validated phase and then to resume transmission at a previous rate without incurring the delay of slow-start. However, if the TCP sender experiences congestion using the preserved cwnd, it is required to immediately reset the cwnd to an appropriate value specified by the method. If a sender does not take advantage of the preserved cwnd within the non-validated period (NVP), the value of cwnd is reduced, ensuring the value better reflects the capacity that was recently actually used.

It is expected that this update will satisfy the requirements of many rate-limited applications and at the same time provide an appropriate method for use in the Internet. New CWV reduces this incentive for an application to send "padding" data simply to keep transport congestion state.

The method is specified in the following subsections and is expected to encourage applications and TCP stacks to use standards-based congestion control methods. It may also encourage the use of long-lived connections where this offers benefit (such as persistent HTTP).

4.1. Initialisation

A sender starts a TCP connection in the validated phase and initialises the pipeACK variable to the "undefined" value. This value inhibits use of the value in cwnd calculations.

4.2. Estimating the Validated Capacity Supported by a Path

[RFC6675] defines "FlightSize", a variable that indicates the instantaneous amount of data that has been sent but not cumulatively acknowledged. In this method, a new variable "pipeACK" is introduced to measure the acknowledged size of the network pipe. This is used

to determine if the sender has validated the cwnd. pipeACK differs from FlightSize in that it is evaluated over a window of acknowledged data, rather than reflecting the amount of data outstanding.

A sender determines a pipeACK sample by measuring the volume of data that was acknowledged by the network over the period of a measured Round-Trip Time (RTT). Using the variables defined in [RFC6675], a value could be measured by caching the value of HighACK and, after one RTT, measuring the difference between the cached HighACK value and the current HighACK value. A sender MAY count TCP DupACKs that acknowledge new data when collecting the pipeACK sample. Other equivalent methods may be used.

A sender is not required to continuously update the pipeACK variable after each received ACK but SHOULD perform a pipeACK sample at least once per RTT when it has sent unacknowledged segments.

The pipeACK variable MAY consider multiple pipeACK samples over the pipeACK Sampling Period. The value of the pipeACK variable MUST NOT exceed the maximum (highest value) within the pipeACK Sampling Period. This specification defines the pipeACK Sampling Period as Max(3*RTT, 1 second). This period enables a sender to compensate for large fluctuations in the sending rate, where there may be pauses in transmission, and allows the pipeACK variable to reflect the largest recently measured pipeACK sample.

When no measurements are available (e.g., a sender that has just started transmission or immediately after loss recovery), the pipeACK variable is set to the "undefined value". This value is used to inhibit entering the non-validated phase until the first new measurement of a pipeACK sample. (Section 4.5 provides examples of implementation.)

The pipeACK variable MUST NOT be updated during TCP Fast Recovery. That is, the sender stops collecting pipeACK samples during loss recovery. The method RECOMMENDS enabling the TCP SACK option [RFC2018] and RECOMMENDS the method defined in [RFC6675] to recover missing segments. This allows the sender to more accurately determine the number of missing bytes during the loss recovery phase, and using this method will result in a more appropriate cwnd following loss.

Note: The use of pipeACK rather than FlightSize can change the behaviour of a TCP flow when a sender does not always have data available to send. One example arises when there is a pause in transmission after sending a sequence of many packets, and the sender experiences loss at or near the end of its transmission sequence. In this case, the TCP flow may have used a significant amount of

capacity just prior to the loss (which would be reflected in the volume of data acknowledged, recorded in the pipeACK variable), but at the actual time of loss, the number of unacknowledged packets in flight (at the end of the sequence) may be small, i.e., there is a small FlightSize. After loss recovery, the sender resets its congestion control state.

[Fai12] explored the benefits of different responses to congestion for application-limited streams. If the response is based only on the Loss FlightSize, the sender would assign a small cwnd and ssthresh, based only on the volume of data sent after the loss. When the sender next starts to transmit, it can incur many RTTs of delay in slow-start before it reacquires its previous rate. When the pipeACK value is also used to calculate the cwnd and ssthresh (as specified in Section 4.4.1), the sender can use a value that also reflects the recently used capacity before the loss. This prevents a variable-rate application from being unduly penalised. When the sender resumes, it starts at one-half its previous rate, similar to the behaviour of a bulk TCP flow [Hos15]. To ensure an appropriate reaction to ongoing congestion, this method requires that the pipeACK variable is reset after it is used in this way.

4.3. Preserving cwnd during a Rate-Limited Period

The updated method creates a new TCP sender phase that captures whether the cwnd reflects a validated or non-validated value. The phases are defined as:

- Validated phase: pipeACK >=(1/2)*cwnd, or pipeACK is undefined (i.e., at the start or directly after loss recovery). This is the normal phase, where cwnd is expected to be an approximate indication of the capacity currently available along the network path, and the standard methods are used to increase cwnd (currently, the standard methods are described in [RFC5681]).

- Non-validated phase: pipeACK <(1/2)*cwnd. This is the phase where the cwnd has a value based on a previous measurement of the available capacity, and the usage of this capacity has not been validated in the pipeACK Sampling Period, that is, when it is not known whether the cwnd reflects the currently available capacity along the network path. The mechanisms to be used in this phase seek to determine a safe value for cwnd and an appropriate reaction to congestion.

Note: A threshold is needed to determine whether a sender is in the validated or non-validated phase. A standard TCP sender in slow-start is permitted to double its FlightSize from one RTT to the next. This motivated the choice of a threshold value of 1/2. This

threshold ensures a sender does not further increase the cwnd as long as the FlightSize is less than (1/2*cwnd). Furthermore, a sender with a FlightSize less than (1/2*cwnd) may, in the next RTT, be permitted by the cwnd to send at a rate that more than doubles the FlightSize; hence, this case needs to be regarded as non-validated, and a sender therefore needs to employ additional mechanisms while in this phase.

4.4. TCP Congestion Control during the Non-validated Phase

A TCP sender implementing this specification MUST enter the non-validated phase when the pipeACK is less than (1/2)*cwnd. (The note at the end of Section 4.4.1 describes why pipeACK<=(1/2)*cwnd is expected to be a safe value.)

A TCP sender that enters the non-validated phase preserves the cwnd (i.e., the cwnd only increases after a sender fully uses the cwnd in this phase; otherwise, the cwnd neither grows nor reduces). The phase is concluded when the sender transmits sufficient data so that pipeACK > (1/2)*cwnd (i.e., the sender is no longer rate-limited) or when the sender receives an indication of congestion.

After a fixed period of time (the non-validated period (NVP)), the sender adjusts the cwnd (Section 4.4.3). The NVP SHOULD NOT exceed five minutes. Section 5 discusses the rationale for choosing a safe value for this period.

The behaviour in the non-validated phase is specified as:

- A sender determines whether to increase the cwnd based upon whether it is cwnd-limited (see Section 4.5.3):
  - A sender that is cwnd-limited MAY use the standard TCP method to increase cwnd (i.e., the standard method permits a TCP sender that fully utilises the cwnd to increase the cwnd each time it receives an ACK).
  - A sender that is not cwnd-limited MUST NOT increase the cwnd when ACK packets are received in this phase (i.e., needs to avoid growing the cwnd when it has not recently sent using the current size of cwnd).
- If the sender receives an indication of congestion while in the non-validated phase (i.e., detects loss), the sender MUST exit the non-validated phase (reducing the cwnd as defined in Section 4.4.1).

- If the Retransmission Timeout (RTO) expires while in the non-validated phase, the sender MUST exit the non-validated phase. It then resumes using the standard TCP RTO mechanism [RFC5681].

- A sender with a pipeACK variable greater than (1/2)*cwnd SHOULD enter the validated phase. (A rate-limited sender will not normally be impacted by whether it is in a validated or non-validated phase, since it will normally not increase FlightSize to use the entire cwnd. However, a change to the validated phase will release the sender from constraints on the growth of cwnd and result in using the standard congestion response.)

The cwnd-limited behaviour may be triggered during a transient condition that occurs when a sender is in the non-validated phase and receives an ACK that acknowledges received data, the cwnd was fully utilised, and more data is awaiting transmission than may be sent with the current cwnd. The sender MAY then use the standard method to increase the cwnd. (Note that if the sender succeeds in sending these new segments, the updated cwnd and pipeACK variables will eventually result in a transition to the validated phase.)

4.4.1. Response to Congestion in the Non-validated Phase

Reception of congestion feedback while in the non-validated phase is interpreted as an indication that it was inappropriate for the sender to use the preserved cwnd. The sender is therefore required to quickly reduce the rate to avoid further congestion. Since the cwnd does not have a validated value, a new cwnd value needs to be selected based on the utilised rate.

A sender that detects a packet drop MUST record the current FlightSize in the variable LossFlightSize and MUST calculate a safe cwnd for loss recovery using the method below:

```
cwnd = (Max(pipeACK,LossFlightSize))/2.
```

The pipeACK value is not updated during loss recovery (see Section 4.2). If there is a valid pipeACK value, the new cwnd is adjusted to reflect that a non-validated cwnd may be larger than the actual FlightSize or recently used FlightSize (recorded in pipeACK). The updated cwnd therefore prevents overshoot by a sender, significantly increasing its transmission rate during the recovery period.

At the end of the recovery phase, the TCP sender MUST reset the cwnd using the method below:

```
cwnd = (Max(pipeACK,LossFlightSize) - R)/2.
```

Where R is the volume of data that was successfully retransmitted during the recovery phase. This corresponds to segments retransmitted and considered lost by the pipe estimation algorithm at the end of recovery. It does not include the additional cost of multiple retransmission of the same data. The loss of segments indicates that the path capacity was exceeded by at least R; hence, the calculated cwnd is reduced by at least R before the window is halved.

The calculated cwnd value MUST NOT be reduced below 1 TCP Maximum Segment Size (MSS).

After completing the loss recovery phase, the sender MUST re-initialise the pipeACK variable to the "undefined" value. This ensures that standard TCP methods are used immediately after completing loss recovery until a new pipeACK value can be determined.

The ssthresh is adjusted using the standard TCP method (Step 6 in Section 3.2 of RFC 5681 assigns the ssthresh a value equal to cwnd at the end of the loss recovery).

Note: The adjustment by reducing cwnd by the volume of data not sent (R) follows the method proposed for Jump Start [Liu07]. The inclusion of the term R makes the adjustment more conservative than standard TCP. This is required, since a sender in the non-validated phase is allowed a rate higher than a standard TCP sender would have achieved in the last RTT (i.e., to have more than doubled the number of segments in flight relative to what was sent in the previous RTT). The additional reduction after congestion is beneficial when the LossFlightSize has significantly overshot the available path capacity, incurring significant loss (e.g., following a change of path characteristics or when additional traffic has taken a larger share of the network bottleneck during a period when the sender transmits less).

Note: The pipeACK value is only valid during a non-validated phase; therefore, this does not exceed cwnd/2. If LossFlightSize and R were small, then this can result in the final cwnd after loss recovery being at most one-quarter of the cwnd on detection of congestion. This reduction is conservative, and pipeACK is then reset to undefined; hence, cwnd updates after a congestion event do not depend upon the pipeACK history before congestion was detected.

4.4.2. Sender Burst Control during the Non-validated Phase

TCP congestion control allows a sender to accumulate a cwnd that would allow it to send a burst of segments with a total size up to the difference between the FlightSize and cwnd. Such bursts can impact other flows that share a network bottleneck and/or may induce congestion when buffering is limited.

Various methods have been proposed to control the sender burstiness [Hug01] [All05]. For example, TCP can limit the number of new segments it sends per received ACK. This is effective when a flow of ACKs is received but cannot be used to control a sender that has not sent appreciable data in the previous RTT [All05].

This document recommends using a method to avoid line-rate bursts after an idle or rate-limited interval when there is less reliable information about the capacity of the network path. A TCP sender in the non-validated phase SHOULD control the maximum burst size, e.g., using a rate-based pacing algorithm in which a sender paces out the cwnd over its estimate of the RTT, or some other method, to prevent many segments being transmitted contiguously at line-rate. The most appropriate method(s) to implement pacing depend on the design of the TCP/IP stack, speed of interface, and whether hardware support (such as TSO) is used. This document does not recommend any specific method.

4.4.3. Adjustment at the End of the Non-validated Period (NVP)

An application that remains in the non-validated phase for a period greater than the NVP is required to adjust its congestion control state. If the sender exits the non-validated phase after this period, it MUST update the ssthresh:

```
ssthresh = max(ssthresh, 3*cwnd/4).
```

(This adjustment of ssthresh ensures that the sender records that it has safely sustained the present rate. The change is beneficial to rate-limited flows that encounter occasional congestion and could otherwise suffer an unwanted additional delay in recovering the sending rate.)

The sender MUST then update cwnd to be not greater than:

```
cwnd = max((1/2)*cwnd, IW).
```

Where IW is the appropriate TCP initial window used by the TCP sender (see, e.g., [RFC5681]).

Note: These cwnd and ssthresh adjustments cause the sender to enter slow-start (since ssthresh > cwnd). This adjustment ensures that the sender responds conservatively after remaining in the non-validated phase for more than the non-validated period. In this case, it reduces the cwnd by a factor of two from the preserved value. This adjustment is helpful when flows accumulate but do not use a large cwnd; this adjustment seeks to mitigate the impact when these flows later resume transmission. This could, for instance, mitigate the impact if multiple high-rate application flows were to become idle over an extended period of time and then were simultaneously awakened by an external event.

4.5. Examples of Implementation

This section provides informative examples of implementation methods. Implementations may choose to use other methods that comply with the normative requirements.

4.5.1. Implementing the pipeACK Measurement

A pipeACK sample may be measured once each RTT. This reduces the sender processing burden for calculating after each acknowledgment and also reduces storage requirements at the sender.

Since application behaviour can be bursty using CWV, it may be desirable to implement a maximum filter to accumulate the measured values so that the pipeACK variable records the largest pipeACK sample within the pipeACK Sampling Period. One simple way to implement this is to divide the pipeACK Sampling Period into several (e.g., five) equal-length measurement periods. The sender then records the start time for each measurement period and the highest measured pipeACK sample. At the end of the measurement period, any measurement(s) that is older than the pipeACK Sampling Period is discarded. The pipeACK variable is then assigned the largest of the set of the highest measured values.

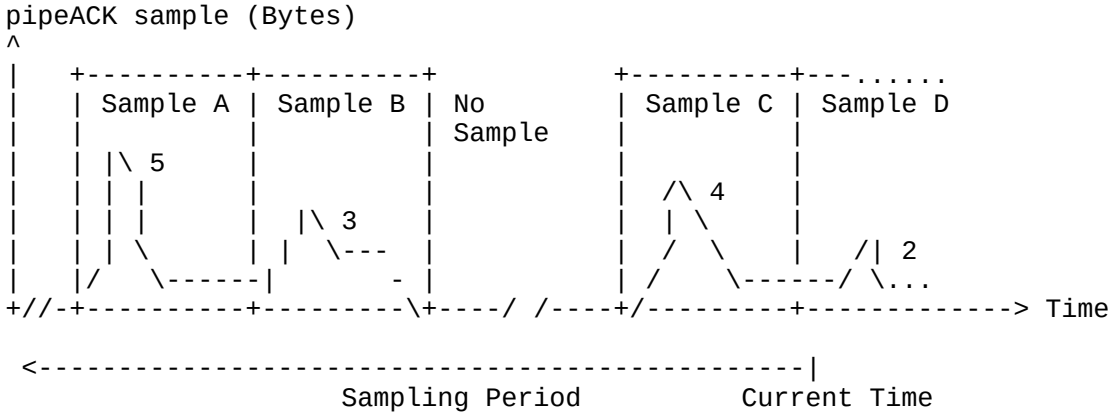

Figure 1: Example of Measuring pipeACK Samples

Figure 1 shows an example of how measurement samples may be collected. At the time represented by the figure, new samples are being accumulated into sample D. Three previous samples also fall within the pipeACK Sampling Period: A, B, and C. There was also a period of inactivity between samples B and C during which no measurements were taken (because no new data segments were acknowledged). The current value of the pipeACK variable will be 5, the maximum across all samples. During this period, the pipeACK samples may be regarded as zero and hence do not contribute to the calculated pipeACK value.

After one further measurement period, Sample A will be discarded, since it then is older than the pipeACK Sampling Period, and the pipeACK variable will be recalculated. Its value will be the larger of Sample C or the final value accumulated in Sample D.

4.5.2. Measurement of the NVP and pipeACK Samples

The mechanism requires a number of measurements of time. These measurements could be implemented using protocol timers but do not necessarily require a new timer to be implemented. Avoiding the use of dedicated timers can save operating system resources, especially when there may be large numbers of TCP flows.

The NVP could be measured by recording a timestamp when the sender enters the non-validated phase. Each time a sender transmits a new segment, this timestamp can be used to determine if the NVP has expired. If the measured period exceeds the NVP, the sender can then take into account how many units of the NVP have passed and make one reduction (defined in Section 4.4.3) for each NVP.

Similarly, the time measurements for collecting pipeACK samples and determining the pipeACK Sampling Period could be derived by using a timestamp to record when each sample was measured and using this to calculate how much time has passed when each new ACK is received.

4.5.3. Implementing Detection of the cwnd-Limited Condition

A sender needs to implement a method that detects the cwnd-limited condition (see Section 4.4). This detects a condition where a sender in the non-validated phase receives an ACK, but the size of cwnd prevents sending more new data.

In simple terms, this condition is true only when the FlightSize of a TCP sender is equal to or larger than the current cwnd. However, an implementation also needs to consider constraints on the way in which the cwnd variable can be used; for instance, implementations need to support other TCP methods such as the Nagle Algorithm and TCP Segment Offload (TSO) that also use cwnd to control transmission. These other methods can result in a sender becoming cwnd-limited when the cwnd is nearly, rather than completely, equal to the FlightSize.

5. Determining a Safe Period to Preserve cwnd

This section documents the rationale for selecting the maximum period that cwnd may be preserved, known as the NVP.

Limiting the period that cwnd may be preserved avoids undesirable side effects that would result if the cwnd were to be kept unnecessarily high for an arbitrarily long period, which was a part of the problem that CWV originally attempted to address. The period a sender may safely preserve the cwnd is a function of the period that a network path is expected to sustain the capacity reflected by cwnd. There is no ideal choice for this time.

A period of five minutes was chosen for this NVP. This is a compromise that was larger than the idle intervals of common applications but not sufficiently larger than the period for which the capacity of an Internet path may commonly be regarded as stable. The capacity of wired networks is usually relatively stable for periods of several minutes, and that load stability increases with the capacity. This suggests that cwnd may be preserved for at least a few minutes.

There are cases where the TCP throughput exhibits significant variability over a time less than five minutes. Examples could include wireless topologies, where TCP rate variations may fluctuate on the order of a few seconds as a consequence of medium access protocol instabilities. Mobility changes may also impact TCP

performance over short time scales. Senders that observe such rapid changes in the path characteristic may also experience increased congestion with the new method; however, such variation would likely also impact TCP's behaviour when supporting interactive and bulk applications.

Routing algorithms may change the network path that is used by a transport. Although a change of path can in turn disrupt the RTT measurement and may result in a change of the capacity available to a TCP connection, we assume these path changes do not usually occur frequently (compared to a time frame of a few minutes).

The value of five minutes is therefore expected to be sufficient for most current applications. Simulation studies (e.g., [Bis11]) also suggest that for many practical applications, the performance using this value will not be significantly different from that observed using a non-standard method that does not reset the cwnd after idle.

Finally, other TCP sender mechanisms have used a five-minute timer, and there could be simplifications in some implementations by reusing the same interval. TCP defines a default user timeout of five minutes [RFC793], which is how long transmitted data may remain unacknowledged before a connection is forcefully closed.

6. Security Considerations

General security considerations concerning TCP congestion control are discussed in [RFC5681]. This document describes an algorithm that updates one aspect of the congestion control procedures, so the considerations described in [RFC5681] also apply to this algorithm.

7. References

7.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <http://www.rfc-editor.org/info/rfc2018>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, DOI 10.17487/RFC2861, June 2000, <http://www.rfc-editor.org/info/rfc2861>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <http://www.rfc-editor.org/info/rfc6298>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <http://www.rfc-editor.org/info/rfc6675>.

7.2. Informative References

[All05] Allman, M. and E. Blanton, "Notes on Burst Mitigation for Transport Protocols", ACM SIGCOMM Computer Communication Review, Volume 35, Issue 2, DOI 10.1145/1064413.1064419, April 2005.

[Bis08] Biswas, I. and G. Fairhurst, "A Practical Evaluation of Congestion Window Validation Behaviour", 9th Annual Postgraduate Symposium in the Convergence of Telecommunications, Networking and Broadcasting (PGNet), Liverpool, UK, 2008.

[Bis10] Biswas, I., Sathiaseelan, A., Secchi, R., and G. Fairhurst, "Analysing TCP for Bursty Traffic", Int'l J. of Communications, Network and System Sciences, DOI 10.4236/ijcns.2010.37078, July 2010.

[Bis11] Biswas, I., "Internet Congestion Control for Variable-Rate TCP Traffic", PhD Thesis, School of Engineering, University of Aberdeen, 2011.

[Fai12] Sathiaseelan, A., Secchi, R., Fairhurst, G., and I. Biswas, "Enhancing TCP Performance to support Variable-Rate Traffic", 2nd Capacity Sharing Workshop, ACM CoNEXT, Nice, France, December 2012.

[Hos15] Hossain, Z., "A Study of Mechanisms to Support Variable-Rate Internet Applications over a Multi-service Satellite Platform", PhD Thesis, School of Engineering, University of Aberdeen, January 2015.

[Hug01] Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP Slow-Start Restart After Idle", Work in Progress, draft-hughes-restart-00, December 2001.

[Liu07] Liu, D., Allman, M., Jin, S., and L. Wang, "Congestion Control without a Startup Phase", 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, California, February 2007.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Acknowledgments

This document was produced by the TCP Maintenance and Minor Extensions (tcpm) working group.

The authors acknowledge the contributions of Dr. I. Biswas and Dr. Ziaul Hossain in supporting the evaluation of CWV and for their help in developing the mechanisms proposed in this document. We also acknowledge comments received from the Internet Congestion Control Research Group, in particular Yuchung Cheng, Mirja Kuehlewind, Joe Touch, and Mark Allman. This work was partly funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
United Kingdom

Email: gorry@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk

Arjuna Sathiaseelan
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
United Kingdom

Email: arjuna@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk

Raffaello Secchi
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
United Kingdom

Email: raffaello@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk